

# Informatik I Übung, Woche 41

Giuseppe Accaputo

8. Oktober, 2015

## Plan für heute

1. Fragen & Nachbesprechung Übung 3
2. Zusammenfassung der bisherigen Vorlesungsslides
3. Tipps zur Übung 4

## Aufgabe 3.1: Präzedenz der Boole'schen Operatoren

```
a = true; b = 1; c = 2;  
IF b > c or a THEN ...
```

⇒ **Syntaxfehler**: Compiler interpretiert die **IF**-Abfrage als **IF b > (c or a)** und gibt einen Fehler aus, da **or** zwei **BOOLEAN** Parameter verlangt, jedoch der Parameter **c** vom Typ **INTEGER** ist

⇒ **Korrekt**:

```
IF (b > c) or a THEN ...
```

- ▶ **Tipp**: Verwende Klammern sobald Vergleichsoperatoren, Boole'sche Operatoren, Zahlen (**REAL**, **INTEGER**, etc.) und **BOOLEAN** Variablen in einer Bedingung vorkommen

## Aufgabe 3.2.c: Programme vereinfachen

prog3.pas:

```
...  
  i := 1;  
  
  WHILE i <= n DO  
  BEGIN  
    IF i mod 2 = 0 THEN  
      g := g + i;  
    Inc(i);  
  END;  
...
```

⇒  $i = 1, \dots, n$ , d.h. Anzahl Iterationen sind bekannt, daher kann eine **FOR**-Schleife verwendet werden

## Bits und Bytes

- ▶ Zahlen werden im Computer als  $n$ -stellige Binärzahlen dargestellt
- ▶ 1 Bit kann Wert 0 oder 1 annehmen
- ▶ 1 Byte besteht aus 8 Bits
- ▶ 1 Word beschreibt die Grösse einer Speicherzelle (8, 16, 32, 64 Bits)

## Darstellung von Dezimalzahlen im Binärsystem

Beispiel: Dezimalzahlen als 8-stellige Binärzahlen darstellen:

Wertigkeit	128	64	32	16	8	4	2	1
94 binär =	0	1	0	1	1	1	1	0

$$\implies 94 = 64 + 16 + 8 + 4 + 2$$

- ▶ 8-stellige Binärzahl  $\implies$  der grösste Wert in der Binärzahl ist  $2^8 = 128$ 
  - ▶ 8. Stelle der Binärzahl repräsentiert  $2^8$
  - 7. Position der Binärzahl repräsentiert  $2^7$
  - ...
  - 1. Stelle der Binärzahl repräsentiert  $2^0 = 1$

## Darstellung von negativen Dezimalzahlen im Binärsystem

- ▶ Zweierkomplement verwenden, um negative Zahlen im Binärsystem darzustellen
  - ▶  $-x$  wird so dargestellt, dass  $-x + x = 0$
- ▶ Vorgehen:
  1. Zahl ohne Vorzeichen ins Binärsystem umrechnen
  2. alle Bits flippen ( $0 \rightarrow 1$  und  $1 \rightarrow 0$ )
  3. 1 addieren
- ▶ Beispiel:  $-4$ 
  1.  $4 = 00000100$
  2. Alle Bits flippen: **not**  $00000100 = 11111011$
  3. 1 addieren:  $11111011 + 00000001 = 11111100$
  4. Kontrolle:  
 $11111100 = -128 + 64 + 32 + 16 + 8 + 4 = -4$  ✓  
oder  $11111100 + 00000100 = 0$  ✓

## Repräsentation von Dezimalzahlen

- ▶ Zwei Arten von Datentypen um Dezimalzahlen darzustellen: Unsigned und Signed
  - ▶ **Unsigned:** nur positive Werte werden dargestellt
  - ▶ **Signed:** positive und negative Werte können dargestellt werden
    - ▶ Binärzahlen, die eine Dezimalzahl durch einen *signed* Datentypen darstellen und als erste Ziffer eine 1 haben sind negativ
- ▶ Beispiel: 11010101
  - ▶ Unsigned:  
 $11010101 = 128 + 64 + 16 + 4 + 1 = 213$
  - ▶ Signed:  
 $11010101 = -128 + 64 + 16 + 4 + 1 = -43$



## Dezimale Gleitkommazahlen

- ▶  $l_1$  Stellen für Mantisse
- ▶  $l_2$  Stellen für Exponent

Beispiel mit  $l_1 = 6$ ,  $l_2 = 2$ :

$$\underbrace{2.99792}_{\text{6 Stellen für Mantisse}} \cdot \underbrace{10^{08}}_{\text{2 Stellen für Exponenten (08)}} \quad (1)$$

2	9	9	7	9	2	0	8
---	---	---	---	---	---	---	---

## Dezimale Gleitkommazahlen: Normalisierung

- ▶ Normalisierte Gleitkommazahl: Dezimale Gleitkommazahl wird so dargestellt, dass die kompletten  $l_1$  Stellen für die Mantisse ausgefüllt werden

Denormalisiert:

<b>0</b>	2	9	9	7	9	<b> </b>	0	9
----------	---	---	---	---	---	----------	---	---

- ▶ Nur 5 der 6 Stellen für die Mantisse werden verwendet bei der denormalisierten Darstellung der Gleitkommazahl

Normalisiert:

2	9	9	7	9	2	<b> </b>	0	<b>8</b>
---	---	---	---	---	---	----------	---	----------

- ▶ Verkleinere den Exponenten um die Gleitkommazahl zu normalisieren. Alle 6 Stellen der Mantisse werden nun für die Darstellung verwendet.

## Überläufe (Overflows) (1/3)

```
VAR a : INTEGER;  
  
BEGIN  
    a := 39364;  
    Writeln(a);  
END.
```

⇒ Ausgabe auf der Konsole ist  $-26172$ . Warum?

## Überläufe (Overflows) (2/3)

**Problem:** Free Pascal verwendet für **INTEGER** automatisch **SMALLINT**, jedoch ist der Wertebereich von **SMALLINT** zu klein um 39364 korrekt darzustellen (Wertebereich:  $[-32768, 32767]$ )

- ▶ 39364 als **SMALLINT** (16-stellige Binärzahl, 16 Bits):  
1001100111000100    **x**  
⇒ 39364 als **SMALLINT** resultiert in  $-26172$
- ▶ 39364 als **LONGINT** (32-stellige Binärzahl, 32 Bits):  
00000000000000001001100111000100    **✓**

## Überläufe (Overflows) (3/3)

Verwende von nun an in jedem Programm folgenden Code-Block:

```
PROGRAM ...  
  
{ $OverflowChecks ON }  
{ $RangeChecks ON }  
TYPE INTEGER = LONGINT;  
  
VAR ...
```

- ▶ Compiler überprüft nun, ob Überläufe entstehen und gibt eine Fehlermeldung aus
- ▶ **INTEGER** ist nun immer ein 32-Bit LONGINT

## DOs and DON'Ts: Addiere zuerst kleine Zahlen

$$10^{16} + 1 + \dots + 1 \neq 1 + \dots 1 + 10^{16}$$

- ▶ Addiert man zuerst die grossen Zahlen und dann die kleinen Zahlen, so rationalisiert der Compiler dies so, dass die kleine Zahlen keinen signifikanten Beitrag leisten und ignoriert diese